ELSEVIER
Discrete Applied Mathematics 49 (1994) 309–323
DISCRETE APPLIED MATHEMATICS

An O(*nm*)-time algorithm for computing the dual of a regular Boolean function

Uri N. Peled*,[a], Bruno Simeone**,[b]

[a] *Mathematics, Statistics, and Computer Science Department, M/C 249, University of Illinois at Chicago, 851 S. Morgan St., Chicago, IL 60607-7045, USA*
[b] *Department of Statistics, University of Rome "La Sapienza", Rome, Italy*

Received 20 March 1991; revised 13 January 1992

Abstract

We consider the problem of *dualizing* a positive Boolean function $f: B^n \to B$ given in irredundant disjunctive normal form (DNF), that is, obtaining the irredundant DNF form of its dual $f^d(x) = \bar{f}(\bar{x})$. The function f is said to be *regular* if there is a linear order $\gtrsim$ on $\{1, \ldots, n\}$ such that $i \gtrsim j$, $x_i = 0$, and $x_j = 1$ imply $f(x) \leqslant f(x + u_i - u_j)$, where u_k denote unit vectors. A previous algorithm of the authors, the Hop-Skip-and-Jump algorithm, dualizes a regular function in polynomial time. We use this algorithm to give an explicit expression for the irredundant DNF of f^d in terms of the one for f. We show that if the irredundant DNF for f has $m \geqslant 2$ terms, then the one for f^d has at most $(n-2)m+1$, and can be computed in O(*nm*) time. This can be applied to solve regular set-covering problems in O(*nm*) time.

1. Introduction

Given a positive Boolean function of n variables in disjunctive normal form (DNF)

$$f(x) = \bigvee_{k=1}^{m} \bigwedge_{j \in S_k} x_j, \tag{1}$$

where $\{S_1, \ldots, S_m\}$ is a collection of subsets of the set $N = \{1, \ldots, n\}$ such that $S_i \not\subseteq S_j$ for all $i \neq j$ (i.e., the DNF is irredundant), the *dual* function f^d of f is defined by

$$f^d(x) = \bar{f}(\bar{x}) = \bigwedge_{k=1}^{m} \bigvee_{j \in S_k} x_j. \tag{2}$$

* Corresponding author. E-mail: U32799@uicvm.uic.edu.
** E-mail: marsalis@itcaspur.bitnet.

SSDI 0166-218X(92)00141-5

Notice that expression (2) is in conjunctive normal form (CNF). However, in many applications such as threshold logic, reliability theory, game theory and combinatorial optimization, one is interested in finding the (unique) irredundant DNF of f^d. Let

$$f^d(x) = \bigvee_{k=1}^{q} \bigwedge_{j \in T_k} x_j \tag{3}$$

be that DNF. An element x of $B^n = \{0, 1\}^n$—henceforth called *a point*—is said to be *feasible* if $f(x) = 0$ and *infeasible* if $f(x) = 1$. A simple but important fact is that the maximal feasible points (MFP's) of f are the characteristic vectors of the complements of the sets T_k in (3), whereas the minimal infeasible points (MIP's) of f are the characteristic vectors of the sets S_k in (1). Here "maximal" and "minimal" refer to the componentwise or product partial order $\leqslant$ on B^n.

In principle, one can always obtain the DNF (3) from the CNF (2) using distributivity, but this may require exponential time. Under stronger assumptions on f, such as regularity, one can compute a DNF for f^d in an efficient way.

The positive Boolean function f is said to be *regular* if there is a linear order $\gtrsim$ on N such that, for every vector $x \in B^n$ and all pairs (i, j) with $i \gtrsim j$, $x_i = 0$ and $x_j = 1$, one has

$$f(x) \leqslant f(x + u_i - u_j), \tag{4}$$

where u_k denotes the kth unit vector $(0, \ldots, 0, 1, 0, \ldots, 0)$. For example, every positive threshold function (i.e., the characteristic function of the solutions of a linear inequality with nonnegative coefficients in 0/1 variables) is regular. The notion of regularity was introduced by Winder [9]. Using his results, one can recognize regular Boolean functions in $O(m^2n)$ time.

An efficient procedure for dualizing a regular Boolean function was proposed by Hammer, Peled and Pollatschek [5], who did not analyze the worst-case complexity, but empirically observed that the average-case complexity grows linearly with m.

Subsequently in [8] we devised the so-called "Hop-Skip-and-Jump" dualization algorithm for regular functions—a modification of the above procedure—and proved a polynomial-time bound, namely $O(n^3m)$, for its running time. Furthermore, we proved that the number q of MFP's is bounded above by $nm + m + n$. Making use of these results and of Karmarkar's linear programming algorithm [7], we obtained the first polynomial-time algorithm for threshold synthesis. Another consequence was that regular set-covering problems can be solved in polynomial time. Later on Crama [4] described a different and elegant dualization algorithm for regular functions running in $O(n^2m)$ time, and obtained an improved upper bound for q, namely $q \leqslant nm$. Bertolazzi and Sassano [2], using the terminology of clutters instead of the one for positive Boolean functions, introduced the class of ideal clutters as a common generalization of regular clutters and matroidal clutters. They presented algorithms to recognize and dualize an ideal clutter and solve the associated set-covering problem in $O(n^3m^2)$ time. In [3] they specialized these methods to regular clutters to perform these tasks in $O(nm)$ time.

In this paper, we investigate the structure of the set of maximal feasible points of a regular Boolean function through a careful analysis of the Hop-Skip-and-Jump

algorithm. Our main result (Theorem 3.3) provides explicit formulas for the MFP's in terms of certain MIP's, that is, an explicit DNF expression of the dual of a regular Boolean function given in DNF. As we shall see, this result has some important consequences:

(1) an $O(nm)$-time implementation of the Hop-Skip-and-Jump algorithm;

(2) an $O(nm)$-time algorithm for regular set covering;

(3) an improved upper bound for the number of MFP's, namely $q \leqslant (n-2)m+1$, when $m \geqslant 2$. Notice that for $m \geqslant 2$, this upper bound is always smaller than nm. Further improvements in the upper bound are likely.

2. The Hop-Skip-and-Jump algorithm

For convenience, the Hop-Skip-and-Jump algorithm will be recalled in this section. For a correctness proof and other details, see [8].

Let f be a regular Boolean function. Without loss of generality we may assume that f is not constant and that the elements of the index set N are numbered so that $1 \gtrsim \cdots \gtrsim n$. The *support* of a point $x \in B^n$ is the set $\mathrm{supp}(x) = \{j \in N: x_j = 1\}$. The *positional representation* of x is the n-vector whose components are the elements of $\mathrm{supp}(x)$ in increasing order, followed by zeros. We say that x' *follows* x and write $x \prec x'$ if the positional representation of x' is lexicographically greater than the positional representation of x. The linear order $\prec$ on B^n will be called the *positional order*. The immediate successor of x in the positional order is denoted $\mathrm{succ}(x)$. Expressions like "before", "between", and "consecutive" always refer to the positional order.

Let us introduce some more notations:

$$b(x) = \begin{cases} 0, & \text{if } x = 0, \\ \max\{j: x_j = 1\}, & \text{if } x \neq 0, \end{cases} \quad \text{(bottom)},$$

$$d(x) = \begin{cases} 1, & \text{if } x = 0 \text{ or } x_1 = \cdots = x_{b(x)} = 1, \\ \max\{j: x_{j-1} = 0,\ x_j = 1\}, & \text{otherwise}, \end{cases}$$

$$\mathrm{fill}(x) = \begin{cases} x + u_{b(x)+1}, & \text{if } b(x) < n, \\ \text{undefined}, & \text{if } b(x) = n, \end{cases}$$

$$\mathrm{brs}(x) = \begin{cases} x - u_{b(x)} + u_{b(x)+1}, & \text{if } b(x) \neq 0, n, \\ \text{undefined}, & \text{if } b(x) \text{ is } 0 \text{ or } n, \end{cases} \quad \text{(bottom right shift)},$$

$$\mathrm{trunc}(x) = x - \sum\{u_j: d(x) \leqslant j \leqslant b(x)\}.$$

For example, if $x = (110011100)$, then:

$b(x) = 7$,

$d(x) = 5$,

$\mathrm{fill}(x) = (110011110)$,

$\mathrm{brs}(x) = (110011010)$,

$\mathrm{trunc}(x) = (110000000)$.

It is easy to see that

$$\mathrm{succ}(x) = \begin{cases} \mathrm{fill}(x), & \text{if } x_n = 0, \\ \mathrm{brs}(x - u_n), & \text{if } x_n = 1. \end{cases} \tag{5}$$

A *shelter* is a MIP s such that brs(s) is either a feasible point or undefined.

The list of shelters, in increasing positional order and followed by a dummy shelter, forms the input of the Hop-Skip-and-Jump algorithm, shown in Fig. 1. An example appears in the next section.

Basically, the algorithm scans all points of B^n in positional order, skipping over large intervals that cannot contain any MFP's. The correctness of the algorithm is expressed by the following theorem:

Theorem 2.1 [8]. *The Hop-Skip-and-Jump algorithm outputs precisely the MFP's in positional order.*

```
s:= first shelter on the list; {s ≠ 0 because f is not constant}
START: x:= 0;
while there are shelters on the list do
{there will always be, at least, the dummy shelter; the algorithm stops in SKIP or JUMP}
  begin {outer while}
    while x ≠ s − u_b(s) do {inner while}
    {if s is the dummy shelter, then x ≠ s − u_b(s) is considered to be true}
      if x_n = 0
        then FILL-UP: x:= fill(x)
        else SKIP:
          begin {skip}
            output x;
            y:= trunc(x);
            if y = 0 then stop else x:= brs(y)
          end {skip}
    {end inner while};
    LEAP:
      begin {leap}
        if s_n = 0
          then HOP: x:= brs(s)
          else JUMP:
            begin {jump}
              output x;
              if s = u_n then stop else x:= succ(s)
            end {jump};
          s:= next shelter on the list
        end {leap}
  end {outer while};
```

Fig. 1. The Hop-Skip-and-Jump algorithm.

3. Properties of the maximal feasible points between two consecutive shelters

In this section we study the structure of the set of the MFP's between two consecutive shelters. One interesting fact is that these MFP's are independent of the remaining shelters.

Definition 3.1. The MFP x is *of the first kind* if $x_n = 1$ and *of the second kind* if $x_n = 0$.

We consider the point x just after START or just after an execution of LEAP. We denote by X the updated versions of x until after the next LEAP is executed. The current shelter is denoted by s and the previous shelter by r (if s is the first shelter, then r is undefined). The following remark, giving x in terms of r, is easily verified by examining JUMP:

Remark 3.2. The point x is given by the formula

$$x = \begin{cases} \mathrm{brs}(r), & \text{if } r_n = 0, \\ \mathrm{brs}(r - u_n), & \text{if } r_n = 1. \end{cases}$$

If r is undefined (because s is the first shelter), then $x = 0$.

We shall describe the MFP's between r and s. To do so, we need the following notations:

$$\sigma = s - u_{b(s)}, \tag{6}$$

$$p = \begin{cases} 0, & \text{if } s \text{ is the dummy shelter,} \\ \min\{j\colon x_j \neq s_j\}, & \text{otherwise,} \end{cases} \tag{7}$$

$$S = \{j\colon p < j \leqslant n,\ x_j = 1\}. \tag{8}$$

If $S \neq \emptyset$, we use the notations:

$$S = \{j_1, \ldots, j_h\} \quad \text{where } j_h < \cdots < j_1, \tag{9}$$

$$x^{(1)} = x + \sum\{u_j\colon j_1 < j \leqslant n\}, \tag{10}$$

$$x^{(k)} = x - u_{j_k} + \sum\{u_j\colon j_k < j \leqslant n,\ x_j = 0\}, \quad k = 2, \ldots, h. \tag{11}$$

The main result of this section is the following theorem:

Theorem 3.3. (1) *There exist MFP's of the first kind between the consecutive shelters r and s if and only if $x_p = 1$ or s is the dummy shelter. In that case $S \neq \emptyset$ and the above MFP's are precisely $x^{(1)}, \ldots, x^{(h)}$, and moreover $x^{(1)} \prec \cdots \prec x^{(h)}$.*

(2) *There exists an MFP of the second kind between r and s if and only if s is not the dummy shelter and $s_n = 1$. In that case this MFP is unique and is equal to $z = s - u_n$, and moreover, if $S \neq \emptyset$, then $x^{(h)} \prec z$.*

Theorem 3.3 is illustrated in Fig. 2.

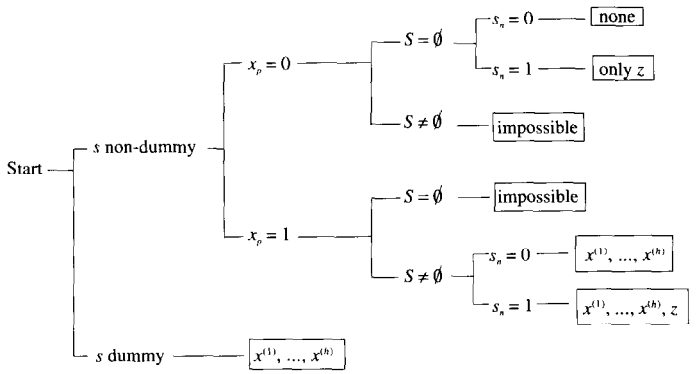

Fig. 2. The MFP's between r and s.

Example. Let the regular Boolean function $f: B^8 \to B$ be defined by

$$f(x) = 0 \quad \Leftrightarrow \quad 27x_1 + 26x_2 + 22x_3 + 9x_4 + 7x_5 + 4x_6 + 3x_7 + 2x_8 \leqslant 60.$$

We find 17 MIP's: 11100000, 11010000, 11001100, 11001010, 11001001, 11000111, 10111000, 10110100, 10110010, 10101110, 10101101, 10101011, 01111000, 01110100, 01110011, 01101110, and 01101101.

Out of these, 9 are shelters (cf. Lemma 4.2): 11010000, 11001001, 11000111, 10110010, 10101101, 10101011, 01110100, 01110011, and 01101101.

These shelters are already in positional order, and they form the input to the algorithm, followed by a dummy shelter. Table 1 describes how the algorithm works on this input, lists p and S every time x changes, and classifies the output according to Fig. 2.

Remark 3.4. Theorem 3.3 completely describes the MFP's between r and s in terms of x and s. In turn, x depends only on r as in Remark 3.2. Therefore these MFP's depend only on r and s, regardless of any other shelters.

We establish Theorem 3.3 by using several lemmas. First assume that s is not the dummy shelter.

Lemma 3.5. *Between r and s, the algorithm outputs zero or more MFP's of the first kind in SKIP. After that it outputs zero or one MFP of the second kind in JUMP. More precisely:*

- *if $s_n = 0$, then no MFP of the second kind is output;*
- *if $s_n = 1$, then exactly one MFP z of the second kind is output, and $z = s - u_n$.*

The algorithm outputs no other MFP's between r and s.

Proof. During the execution of the inner **while**, the algorithm outputs only MFP's of the first kind in SKIP. Then it goes over to LEAP with $X = s - u_{b(s)}$. If $s_n = 0$, then X "HOPs" over s and no output takes place. If $s_n = 1$, then the algorithm outputs

Table 1
An illustration of the algorithm

Step	Shelter s	x	X	MFP output	p	S	Remarks on output
START	11010000	00000000			1	$\emptyset$	
FILL-UPs			11000000				no MFP
HOP	11001001	11001000			8	$\emptyset$	
JUMP				11001000			only z
	11000111	11000100			7	$\emptyset$	
FILL-UP			11000110				
JUMP				11000110			only z
	10110010	11000101			2	$\{6, 8\}$	
SKIP			11000010	11000101			$x^{(1)} = x$
FILL-UP			11000011				
SKIP			10100000	11000011			$x^{(2)}$
FILL-UP			10110000				
HOP	10101101	10110001			4	$\{8\}$	
SKIP			10101000	10110001			$x^{(1)} = x$
FILL-UP			10101100				
JUMP				10101100			z
	10101011	10101010			8	$\emptyset$	
JUMP				10101010			only z
	01110100	10101001			1	$\{3, 5, 8\}$	
SKIP			10100100	10101001			$x^{(1)} = x$
FILL-UPs			10100111				
SKIP			10010000	10100111			$x^{(2)}$
FILL-UPs			10011111				
SKIP			01000000	10011111			$x^{(3)}$
FILL-UPs			01110000				
HOP	01110011	01110010			8	$\emptyset$	
JUMP				01110010			only z
	01101101	01110001			4	$\{8\}$	
SKIP			01101000	01110001			$x^{(1)} = x$
FILL-UP			01101100				
JUMP				01101100			z
	dummy	01101010			0	$\{2, 3, 5, 7\}$	
FILL-UP			01101011				
SKIP			01100100	01101011			$x^{(1)} = x$
FILL-UPs			01100111				
SKIP			01010000	01100111			$x^{(2)}$
FILL-UPs			01011111				
SKIP			00100000	01011111			$x^{(3)}$
FILL-UPs			00111111				
SKIP				00111111			$x^{(4)}$

$X = s - u_n$ as an MFP of the second kind, and X "JUMPs" over s. In both cases $X \succ s$ becomes true, and hence no further MFP's between r and s are output. □

Lemma 3.6. *If $x \not\leqslant s$, then the algorithm performs a sequence of FILL-UPs until X_n becomes* 1.

Proof. By assumption, $x_j = 1$ and $s_j = 0$ for some j. Then throughout the inner **while**, one has $X_j = 1$ and hence $X \neq \sigma$, so no LEAP can occur before X_n becomes 1. □

Lemma 3.7. *One has $x_p = 1$ if and only if $x_{j-1} = 0$ and $x_j = 1$ for some j, $p < j \leqslant n$. In particular, $x_p = 0$ implies $x_{p+1} = \cdots = x_n = 0$.*

Proof. If $x_p = 0$ and $x_j = 1$ for some j, $p < j \leqslant n$, then $s \prec x$, a contradiction to the fact that s is the first shelter following x [8, Lemma 2]. Now assume, if possible, that $x_p = 1$ but no j with the required properties exists. Then $x_j = 1$ for all j, $p \leqslant j \leqslant b(x)$. Since $x_p = 1$ and $s_p = 0$, by Lemma 3.6 the algorithm performs a sequence of FILL-UPs until X_n becomes 1. At this time $X \geqslant s$, a contradiction to the feasibility of X [8, Lemma 2]. □

Lemma 3.8. *Assume that $x_p = 0$ (and thus $b(x) < p \leqslant b(s)$ by Lemma 3.7). Then for each j satisfying $b(x) < j < b(s)$, $s_{j-1} = 0$ implies $s_j = 0$.*

Proof. Assume, if possible, that $s_{j-1} = 0$ but $s_j = 1$. Then $\sigma_{j-1} = 0 < 1 = \sigma_j$. However, $x_{j-1} = x_j = 0$, and therefore the relation $X_{j-1} \geqslant X_j$ is maintained throughout the inner **while**, implying $X \neq \sigma$. See Table 2. Hence the algorithm performs a sequence of FILL-UPs until X_n becomes 1. But at this time $X \geqslant s$, a contradiction. □

Lemma 3.9. *If $x_p = 0$ and $p < b(s)$, then $p = b(x) + 1$.*

Proof. We have $b(x) < p$ by Lemma 3.7. If $b(x) \leqslant p - 2$, then by (7) $s_{b(x)+1} = x_{b(x)+1} = 0$. Since by assumption $b(x) + 2 \leqslant p < b(s)$, the hypothesis of Lemma 3.8 is satisfied for $j = b(x) + 2$. Thus by Lemma 3.8, $s_k = 0$ for $b(x) + 1 \leqslant k < b(s)$. However, $s_p = 1$ and $b(x) + 1 \leqslant p < b(s)$, a contradiction. This shows that $b(x) = p - 1$. □

Lemma 3.10. *If $x_p = 0$, then there is no MFP of the first kind between r and s.*

Proof. We have $p \leqslant b(s)$. Let us distinguish two cases:

Case 1: $p = b(s)$. Then the condition $x = \sigma$ is satisfied before the inner **while** has started.

Case 2: $p < b(s)$. Then $b(x) = p - 1$ by Lemma 3.9. Put $l = \max\{j: p \leqslant j < b(s), s_j = 1\}$. Then $b(x) < l$. See Table 3. The algorithm performs a sequence of FILL-UPs until X_l becomes 1, and then the condition $X = \sigma$ is satisfied.

In either case, the inner **while** results in an X such that $X_n = 0$ without ever performing a SKIP. Then a LEAP is performed, and the result follows from Lemma 3.5. □

Table 2
Illustrating the proof of Lemma 3.8

	$b(x)$	$\cdots$	$j-1$	j	$\cdots$	$b(s)$
x:	1		0	0	$\cdots$	0
s:	1		0	1		1
σ:	1		0	1		0

Table 3
Illustrating the proof of Lemma 3.10

	$b(x)=p-1$	p	$\cdots$	l	$\cdots$	$b(s)$
x:	1	0	$\cdots$	0	0 $\cdots$ 0	0
s:	1	1	$\cdots$	1	0 $\cdots$ 0	1

Table 4
Illustrating the proof of Lemma 3.11

	$b(\sigma)$	$\cdots$	p	$\cdots$	$b(s)$
x:	1		1		
s:	1	0 $\cdots$ 0	0 0 $\cdots$ 0		1
σ:	1	0 $\cdots$ 0	0 0 $\cdots$ 0		0

Lemma 3.11. *If $x_p = 1$, then $p < b(\sigma)$ and $s_j = 1$ for all j, $p < j \leqslant b(\sigma)$.*

Proof. By Lemma 3.7, $p < b(s)$. One cannot have $p = b(\sigma)$, because $s_p = 0$ whereas $s_{b(\sigma)} = \sigma_{b(\sigma)} = 1$. Assume, if possible, that $b(\sigma) < p$. Then $s_j = \sigma_j = 0$ for all j such that $b(\sigma) < j < b(s)$, and $x_{b(\sigma)} = s_{b(\sigma)} = 1$. See Table 4. If $x_{b(s)} = 1$, then $x \geqslant s$, a contradiction. If $x_{b(s)} = 0$, then $x - u_p + u_{b(s)}$ is still feasible and lies above s, a contradiction. Hence $p < b(\sigma)$.

Now assume, if possible, that $s_{j^*} = 0$ for some j^* with $p < j^* < b(\sigma)$. Consider the point X after the SKIP that outputs $x^{(h)}$. At this time $X_p = 0$, $X_{p+1} = 1$, $X_j = 0$ for all $p + 1 < j \leqslant n$, and $X_{b(\sigma)} = 0 \neq 1 = \sigma_{b(\sigma)}$. Hence the algorithm proceeds to FILL-UP X and keeps doing this as long as $X_{b(\sigma)}$ remains 0. Before $X_{b(\sigma)}$ becomes 1, X_{j^*} must take the value 1, and since $s_{j^*} = 0$, the algorithm continues to FILL-UP X until X_n becomes 1 by Lemma 3.6. But then the resulting X lies above s, a contradiction. □

Lemma 3.12. *If $x_p = 1$, then $S \neq \emptyset$ and the points $x^{(1)}, x^{(2)}, \ldots, x^{(h)}$ given by (9)–(11) are precisely the MFP's of the first kind between r and s, and they are output by the algorithm in this order.*

Proof. By Lemma 3.7, the set S is nonempty. As long as X_p remains 1, the algorithm never leaves the inner **while** by Lemma 3.6. More specifically, the algorithm does the following:

(1) starting from x, it successively FILLs-UP the positions $j_1 + 1, \ldots, n$;
(2) it outputs $x^{(1)}$ in SKIP;
(3) it truncates $x^{(1)}$ and performs a bottom right shift from position j_2; at this time $b(X) = j_2$;
(4) it successively FILLs-UP the positions $j_2 + 1, \ldots, n$;
(5) it outputs $x^{(2)}$ in SKIP;

and so on until $x^{(h)}$ has been output. Subsequently X is updated in SKIP. By Lemma 3.11, X continues to be FILLed-UP until $X_{b(\sigma)}$ becomes 1. At that time $X = \sigma$, a LEAP is performed, and $X \succ s$ becomes true, so there are no MFP's of the first kind other than $x^{(1)}, \ldots, x^{(h)}$ between r and s. □

The case that s is the dummy (last) shelter is very similar to the above analysis, as shown by the following lemma:

Lemma 3.13. *If s is the dummy shelter, then the MFP's that are output between r and s are precisely $x^{(1)}, \ldots, x^{(h)}$ given by* (7)–(11) *(with $p = 0$).*

Proof. If s is the dummy shelter, then by convention $x \neq s - u_{b(s)}$ is always true and the algorithm remains in the inner **while**. Furthermore, $p = 0$ and hence $S = \mathrm{supp}(x)$, which is nonempty because $x \neq 0$ (as x follows the previous shelter). The rest of the proof is similar to the proof of Lemma 3.12. □

Proof of Theorem 3.3. If $x_p = 0$, then there is no MFP of the first kind between r and s by Lemma 3.10.

If $x_p = 1$, then by Lemma 3.12 there are MFP's of the first kind and they are given by (7)–(11).

If x_p is undefined, then p is 0 and s is the dummy shelter. In this case, by Lemma 3.13, there are MFP's of the first kind, and they are given by (7)–(11).

Thus the first part of the theorem is proved.

By Lemma 3.5, if $s_n = 0$, there is no MFP of the second kind, and if $s_n = 1$, then $z = s - u_{b(s)}$ is the only MFP of the second kind. Hence the theorem follows. □

Corollary 3.14. *If t is any MFP such that $r \prec t \prec s$, then $t_p = t_n$.*

Corollary 3.15. *The following recursion holds for the MFP's of the first kind between r and s:*

$$x^{(1)} = x + \sum\{u_j : j_1 < j\}, \tag{12}$$

$$x^{(2)} = x^{(1)} - u_{j_2} + \sum\{u_j : j_2 < j < j_1\}, \tag{13}$$

$$x^{(k)} = x^{(k-1)} - u_{j_k} + \sum\{u_j : j_k < j \leqslant j_{k-1}\}, \quad 3 \leqslant k \leqslant h. \tag{14}$$

Proof. Immediate from (10) and (11). □

Remark 3.16. Theorem 3.3 was anticipated in a weaker form, without proofs, and in set-covering terminology, in [6].

Corollary 3.15 is the basis of the $O(mn)$-time implementation of the Hop-Skip-and-Jump algorithm in the next section.

4. The algorithm

Our procedure for generating all the MFP's of the regular Boolean function f given by (1) consists of the same three stages as those in the original procedure of [5] and in

our modification [8], namely:

Stage 1. Sorting the MIP's in positional order;
Stage 2. Extracting the shelters from the sorted list of MIP's;
Stage 3. Generating the MFP's from the sorted list of shelters.

We discuss the implementation of these three stages separately.
Stage 1: Given an arbitrary point $x \in B^n$, let

$$x^{\#} = x + 2\sum\{u_j : j > b(x)\}.$$

Lemma 4.1. *For all $x, y \in B^n$, $x \prec y$ if and only if $x^{\#}$ is lexicographically larger than $y^{\#}$.*

Proof. Follows easily from (5). □

On the basis of Lemma 4.1, we can sort the MIP's x in positional order by sorting the corresponding vectors $x^{\#}$ in reverse lexicographical order. This can be done in $O(nm)$ time using Radix Sort (see e.g. [1]). Lemma 4.1 was stated in [5]. However, the authors did not use Radix Sort and reported an $O(nm \log m)$-time bound for this stage.

Stage 2: As noticed already in [5], one can extract the shelters in positional order from the sorted list of MIP's in $O(nm)$ time by exploiting the following result:

Lemma 4.2 [5]. *Assume that the MIP's are sorted in positional order. Then a MIP s is a shelter if and only if $s_n = 1$ or* brs(s) *does not coincide with the next MIP.*

Stage 3: On the basis of Theorem 3.3 and Corollary 3.15, it is possible to generate all the MFP's between any two consecutive shelters in overall $O(n)$ time. For this purpose if suffices to replace the inner **while** of the Hop-Skip-and-Jump algorithm with the block INNER CYCLE described below.

For convenience let us define, for a given $y \in B^n$ and for given $1 \leqslant i \leqslant j \leqslant n$:

$$\text{ahead}(y; j) = \begin{cases} 0, & \text{if there is no } k < j \text{ such} \\ & \text{that } y_k = 1, \\ \max\{k:\ 1 \leqslant k < j,\ y_k = 1\}, & \text{otherwise,} \end{cases}$$

$$\text{compl}(y; i, j) = y + \sum\{u_k : i < k \leqslant j,\ y_k = 0\}.$$

The block INNER CYCLE is shown in Fig. 3.

In INNER CYCLE, the total time spent on compl is $O(n)$ because the various compl operate on disjoint intervals. Similarly, the total time spent on ahead is $O(n)$. The time for all the rest is $O(n)$ if we disregard output time. Thus the block INNER CYCLE takes $O(n)$ time, and upon its completion the shelter changes. Therefore Stage 3 takes $O(nm)$ time.

In conclusion, each of Stages 1, 2, 3 requires at most $O(nm)$ elementary operations. It follows that the overall complexity of the algorithm is $O(nm)$.

We conclude this section by pointing out that on the previous results, one can solve a regular set-covering problem in $O(nm)$ time for an input matrix of size $m \times n$. One

```
INNER CYCLE:
begin {inner cycle}
  if s is the dummy shelter
    then p := 0
    else p := min{i: x_i ≠ s_i};
  j := b(x);
  if j > p then do
    begin {compute x^(1)}
      x := compl(x; j, n);
      output x;
      i := ahead(x; j);
      if i > p then {compute x^(2)}
        begin
          x := compl(x; i, j − 1) − u_i;
          output x;
          j := i;
          i := ahead(x; j)
        end
      end
    else i := j;
  while i > p do {compute x^(3), ..., x^(h)}
    begin {while}
      x := compl(x; i, j) − u_i;
      output x;
      j := i;
      i := ahead (x; j)
    end {while};
  if s is not the dummy shelter then x := s − u_b(s)
end {inner cycle}
```

Fig. 3. The block INNER CYCLE.

version of the *set-covering problem is*

$$\begin{aligned} &\text{maximize} && cx, \\ &\text{subject to} && Ax \leqslant b, \\ & && x \in B^n, \end{aligned} \tag{15}$$

where A is an $m \times n$ 0/1 matrix, $b = Ae' - e$ and e, e' are all-1 vectors of appropriate dimensions. We assume without loss of generality that c is a vector of positive numbers, that A contains both 0's and 1's, and that no row of A lies above another row. If f is the Boolean function whose feasible points are the feasible solutions of (15), then from the above assumptions f is not constant and its MIP's are the rows of A. If f is regular, we say that the set-covering problem (15) is *regular*. To solve such a problem, we can make use of the following result, which follows easily from Theorem 3.3 and its Corollary 3.15:

Lemma 4.3 [6, Lemma 16]. *Consider the regular set-covering problem* (15) *and the corresponding regular Boolean function f. Let* S, j_k *and* $x^{(k)}$ *be defined as in* (8)–(11) *and*

assume that $h = |S| \geqslant 2$. *Set*

$$\delta_k = \sum\{c_j : j_k < j < j_{k-1}, x_j^{(k)} = 0\},$$

$$\Delta_1 = 0,$$

$$\Delta_2 = \delta_2 - c_{j_2},$$

$$\Delta_k = \Delta_{k-1} + \delta_k + c_{j_{k-1}} - c_{j_k}, \quad 2 < k \leqslant h.$$

Then

$$cx^{(k)} - cx^{(1)} = \Delta_k.$$

As mentioned in [6], Corollary 3.15 and Lemma 4.3 allow one to compute in $O(n)$ time, for each pair of consecutive shelters r and s, the maximum

$$\max\{c_1 y_1 + \cdots + c_n y_n : r \prec y \prec s \text{ and } y \text{ is a MFP}\},$$

and thus to solve the regular set-covering problem (15) in $O(nm)$ time *provided that the sorted list of shelters is available.* The authors there concluded that the overall time complexity for solving the above problem was $O(nm \log m)$—the time required in order to sort the shelters via an adaptation of Heapsort. However, as we have seen above, one can actually compute the sorted list of shelters in $O(nm)$ time. Hence the $O(nm)$ bound for regular set covering follows.

5. An improved upper bound for the number of maximal feasible points

In this section we derive an improved upper bound for the number of maximal feasible points of a regular function. Once more the cornerstone of our analysis is Theorem 3.3. However, in order to obtain this bound, we take a closer look at the structure of the MFP's: our basic strategy is to prove the existence of some nonzero components in the point x (see Lemmas 5.1, 5.2, 5.4 below).

Lemma 5.1. *If* $x \neq 0$, *then* $x_{j_1 - 1} = 0$.

Proof. Since $x \neq 0$, s is not the first shelter. Let r be the previous shelter. The point x is obtained from r as in Remark 3.2, and thus $x_{b(x)-1} = 0$. Since $j_1 = b(x)$, the result follows. □

Lemma 5.2. *If* $x_p = 1$, *then* $x_j = 0$ *for some* j *with* $p < j \leqslant b(\sigma)$.

Proof. Assume the contrary. If $x_{b(s)} = 1$, then $x > s$, a contradiction. If $x_{b(s)} = 0$, then $x - u_p + u_{b(s)} \geqslant s$ and $x - u_p + u_{b(s)}$ is a right shift of x, a contradiction. □

Remark 5.3. Assume that $x_p = 1$. If $b(x) \leqslant b(s)$ (in particular if $s_n = 1$), then there are at least two indices j with $p < j \leqslant b(\sigma)$ such that $x_j = 0$.

Proof. By Lemma 5.2, there is at least one index l with $p < l \leqslant b(\sigma)$ such that $x_l = 0$. Assume, if possible, that l is unique. Since $x_p > s_p$, by Lemma 3.6, the algorithm FILLs-UP x until X_n becomes 1. The FILLed-UP X is feasible and hence $X' = X - u_p + u_l$ is also feasible. However, $X' \geqslant s$ because:

- $X'_j = s_j$ for $j < p$;
- $X'_p = s_p = 0$;
- $X'_j = 1$ for $p < j \leqslant b(\sigma)$;
- $s_j = 0$ for $b(\sigma) < j \leqslant n, j \neq b(s)$;
- $X'_{b(s)} = s_{b(s)} = 1$ ($X'_{b(s)} = 1$ since $b(x) \leqslant b(s)$).

This is a contradiction. □

Lemma 5.4. *If $x_p = 1$ and there is only one l with $p < l \leqslant b(\sigma)$ such that $x_l = 0$, then $x_j = 0$ for all j such that $b(\sigma) < j \leqslant b(s)$.*

Proof. Assume, if possible, that there is a v with $b(\sigma) < v \leqslant b(s)$ such that $x_v = 1$. Then the point $x' = x - u_p + u_l - u_v + u_{b(s)}$ is feasible and $x' \geqslant s$, a contradiction. □

Corollary 5.5. *If $x_p = 1$, there are at least two indices $j > p$ such that $x_j = 0$.*

Proof. Follows from Lemmas 5.2 and 5.4. □

Lemma 5.6. *If $n \geqslant 3$ and s is not the dummy shelter, then there are at most $n - 2$ MFP's between r and s.*

Proof. Since s is not the dummy shelter, $p \geqslant 1$. If $x_p = 0$, then by the second part of Theorem 3.3, there is at most one MFP between r and s. If $x_p = 1$, there are exactly $|S|$ MFP's of the first kind and at most one MFP of the second kind between r and s, again by Theorem 3.3. By Corollary 5.5 and since $p \geqslant 1$, we have $|S| \leqslant n - 3$ and the result follows. □

Lemma 5.7. *There is at most one MFP before the first shelter.*

Proof. At the start, $x = 0$ and hence $x_p = 0$. Hence the result follows from the second part of Theorem 3.3. □

Lemma 5.8. *If s is the dummy shelter and $r_j = 0$ for some $j < n$, then there are at most $n - 2$ MFP's between r and s.*

Proof. By Lemma 5.1, $|S| \leqslant n - 1$, with equality holding if and only if $x = e - u_{n-1}$, where e is the all-1 vector. Assume that indeed $x = e - u_{n-1}$. Then by Remark 3.2, one must have either $r = e$ or $r = e - u_n$. In both cases $r_j = 1$ for all $j < n$, a contradiction. Hence $|S| \leqslant n - 2$, and the result follows from Theorem 3.3. □

Theorem 5.9. *For $m \geqslant 2$, the number q of MFP's satisfies*

$$q \leqslant (n-2)m^* + 1 \leqslant (n-2)m + 1, \tag{16}$$

where m^* is the number of nondummy shelters.

Proof. Since f is not constant, $n \geqslant 2$. Assume that $n = 2$ and (16) fails. Then $q = 2$ and the two MFP's must be (0, 1) and (1, 0). But then the only MIP is (1, 1) and $m = 1$, a contradiction. Now assume that $n \geqslant 3$.

We assert that the last nondummy shelter r is different from both e and $e - u_n$. Indeed, if $r = e$, then obviously $m = 1$, a contradiction. If $r = e - u_n$, then the point $e - u_{n-1} = \mathrm{brs}(r)$ is feasible since r is a shelter, and hence $e - u_j$ is feasible for all $j < n$ by regularity. It follows that e and r are the only infeasible points, hence r is the only MIP and $m = 1$, a contradiction again.

From the assertion, Lemma 5.8 applies when s is the dummy shelter, and therefore there are at most $n - 2$ MFP's after the last nondummy shelter. Moreover, by Lemma 5.6 there are at most $n - 2$ MFP's between consecutive nondummy shelters, and by Lemma 5.7 there is at most one MFP before the first shelter. It follows that

$$q \leqslant 1 + (n-2)(m^* - 1) + (n-2) \leqslant 1 + (n-2)m. \qquad \square$$

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms (Addison-Wesley, Reading, MA, 1974).

[2] P. Bertolazzi and A. Sassano, A class of polynomially solvable set-covering problems, SIAM J. Discrete Math. 1 (1988) 306–316.

[3] P. Bertolazzi and A. Sassano, An $O(mn)$ algorithm for regular set-covering problems, Theoret. Comput. Sci. 54 (1987) 237–247.

[4] Y. Crama, Dualization of regular Boolean functions, Discrete Appl. Math. 16 (1987) 79–85.

[5] P.L. Hammer, U.N. Peled and M.A. Pollatschek, An algorithm to dualize a regular switching function, IEEE Trans. Comput. 28 (1979) 238–243.

[6] P.L. Hammer and B. Simeone, Order relations of variables in 0-1 programming, in: S. Martello, G. Laporte, M. Minoux and C. Ribero, eds., Surveys in Combinatorial Optimization, Annals of Discrete Mathematics 31 (North-Holland, Amsterdam, 1987) 83–112.

[7] N. Karmarkar, A new polynomial algorithm for linear programming, Combinatorica 4 (1984) 373–396.

[8] U.N. Peled and B. Simeone, Polynomial-time algorithms for regular set-covering and threshold synthesis, Discrete Appl. Math. 12 (1985) 57–69.

[9] R.O. Winder, Threshold logic, Ph.D. Dissertation, Department of Mathematics, Princeton University, Princeton, NJ (1962).